ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ЛЕСОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра информационных технологий и моделирования

О.А. Карасева

КОРПОРАТИВНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

Учебно-методические указания
по проведению лабораторно-практических занятий
для студентов дневной, очно-заочной и заочной форм обучения
специальности 080801 – Прикладная информатика (в экономике)
направления 080800 − Прикладная информатика

Часть I

Екатеринбург
2009

Печатаются по рекомендации методической комиссии ФЭУ.
Протокол № 1 от 03.09.2008г.

Рецензент − доцент кафедры ИТМ А.И. Монтиле

Редактор Н.А. Майер
Оператор Г.И. Романова

Подписано в печать 17.04.09		Внеплановая
Плоская печать	Формат 60×84 1/16	Тираж 100 экз.
Заказ №	Печ. л. 2,09	Цена 7 руб. 00 коп.

Редакционно-издательский отдел УГЛТУ
Отдел оперативной полиграфии УГЛТУ

Введение

Данные методические указания (МУ) предназначены для знакомства с основными принципами построения корпоративных информационных систем. В МУ рассматриваются различные архитектуры построения распределенных информационных систем, лежащих в основе корпоративных ИС. Внимание уделяется следующим технологиям:

- репликации данных;
- объектного связывания;
- создания и использования источников данных ODBC.

Рассмотрены провайдеры данных и примеры использования их в приложении, способы подключения объектов доступа к данным DAO, ADODB, OLE DB.

Лабораторная работа 1
Технология реплицирования данных

Цель работы. Знакомство с технологией реплицирования данных средствами Access.

Репликация баз данных применяется для создания специальных “горячих” копий БД средствами Access. С помощью репликации можно легко получать новые копии БД, используемые как на одном компьютере, так и в сети. Отдельные копии (реплики) требуется периодически синхронизировать.

По одной базе данных можно создать набор реплик. В наборе различают основную и дополнительные реплики. Основная реплика отличается от обычной дополнительной реплики тем, что в ней можно изменять структуру БД. Основную реплику можно сделать обычной, а дополнительную − основной, но в любой момент времени в наборе реплик одна реплика является основной, а остальные − дополнительными. Дополнительные реплики можно создавать из основной и из дополнительных реплик.

Создание основной реплики состоит в преобразовании файла исходной БД в новый файл. Исходную базу данных будем называть *реплицируемой*. Если основную реплику назвать тем же именем, что и исходная (реплицируемая) БД, то последняя пропадает. Для безопасности перед проведением преобразования целесообразно создать резервную копию исходного файла БД. Если пользователь не создал резервную копию новой БД до начала реплицирования, он может это сделать в процессе его выполнения.

В ходе репликации в файл исходной БД добавляются специальные таблицы, поля и свойства. После репликации исходная база данных становится основной репликой в наборе реплик. Основная и дополнительная реплики могут содержать реплицируемые и нереплицируемые (локальные) объекты. Полученная основная реплика, с точки зрения работы пользователя, не отличается от исходной БД.

При репликации базы данных Access добавляет системные таблицы, полные имена которых имеют вид: "*nnn.conflict*", где *nnn* представляет имя исходной таблицы. При репликации используются следующие системные таблицы:

- MSysSidclables используется во время неразрешенного при синхронизации конфликта;

- MSysErrors служит для указания места и причины возникновения ошибки при синхронизации данных. В ней указываются: таблица, в которой возникли ошибки; реплика или реплики, в которых также обнаружены ошибки; реплика, в которой строка изменена последней; тип операции, которую не удалось выполнить, и причина;

- MSysSchemaProb появляется в случае возникновения ошибки при обновлении структуры реплики. В ней хранится дополнительная информация о причине ошибки;

- MSysExchangeLog предназначена для хранения информация о выполненных операциях синхронизации реплик.

Изменять содержимое последних трех таблиц пользователю не разрешается. Перечисленные таблицы могут быть видимыми или невидимыми, в зависимости от того, как установлен флажок *Системные объекты* (System Objects) на вкладке *Вид* (View). В каждую таблицу реплицированной БД добавляются следующие системные поля:

• s-GUID − глобальный уникальный идентификатор каждой записи;

• s-Lineage − двоичное поле, содержащее информацию об истории изменений каждой записи;

• s-Generation − поле, содержащее информацию о групповых изменениях.

Реплицироваться могут все объекты БД: таблицы, формы, запросы, отчеты, макросы и модули. В наборе реплик реплицируемыми должны быть одни и те же объекты. Каждая из реплик может содержать свои собственные локальные объекты, структура и содержание которых не передаются в другие реплики.

Синхронизацией называют процесс обновления двух компонентов в наборе реплик, при котором производится обмен обновленными записями и объектами из каждого компонента. *Access* одной командой позволяет выполнить синхронизацию между двумя репликами одного набора. При этом можно синхронизировать между собой две дополнительные реплики.

Проектирование БД с учетом репликации

Как и при создании многопользовательской БД, при проектировании БД с учетом репликации следует иметь в виду несколько важных моментов. Это не значит, что БД, первоначально предназначенную для работы на одном компьютере, нельзя конвертировать в реплицируемую БД, но заблаговременное планирование позволяет выполнить такое преобразование более гладко. Прежде всего следует обратить внимание на первичный ключ таблицы, особенно если он основан на полях типа *Счетчик*. По умолчанию нумерация в таких полях начинается с единицы и последовательно возрастает для каждой следующей записи. Поэтому если несколько пользовате-

лей введут записи в репликах БД, а потом попытаются синхронизировать эти реплики, то может возникнуть ситуация, когда несколько различных записей будут иметь одинаковые значения первичного ключа. Для того чтобы избежать появление такой путаницы, первичные ключи со счетчиком следует создавать согласно двум правилам:

1. Указать значение *Случайные* для свойства *Новые значения*.

2. Указать значение *Код репликации* для свойства *Размер поля* первичного ключа.

Хотя для хранения данных типа *Код репликации* требуется больше места, чем для типа данных *Последовательные*, при создании реплицируемых таблиц рекомендуется использовать именно данные типа *Код репликации*. Их применение практически исключает возможность появления конфликтных ситуаций с записями при выполнении репликации. Однако на такие данные не очень удобно ссылаться, так как они выражаются в шестнадцатеричном формате. Если первичный ключ необходимо использовать для двойного назначения (например, дополнительно для указания номера счета или идентификации сотрудника), то в таком случае в качестве ключа рекомендуется использовать значения *Случайные* типа *Длинное целое*.

Задания к лабораторной работе 1

1. Выполните репликацию вашей базы данных (расположение основной и дополнительной реплик выберите сами).

Репликацию базы данных Access можно выполнять по команде *Сервис, Репликация* меню Access.

Перед созданием основной реплики рекомендуется создать резервную копию исходной БД, так как последняя при реплицировании будет преобразована. Для создания реплики нужно открыть исходную БД и выдать команду *Сервис*, *Репликация, Создать дополнительную реплику*.

Система Access выдаст предупреждающее сообщение о закрытии БД. В очередном окне предлагается выбрать вариант дальнейших действий по репликации БД: создавать / не создавать резервную копию исходной базы данных или отменить репликацию. Если копия уже создана, следует нажать кнопку *Нет*. Автоматически создаваемая резервная копия исходной БД (если пользователь не изменит ее имя) хранится в той же папке, где и основная БД. Имя файла резервной копии (если пользователь его не изменит) совпадает с именем файла исходной БД, а имя файла имеет расширение *bak*.

В очередном окне остается определить имя и местоположение основной реплики.

Для создания дополнительной реплики достаточно открыть основную или дополнительную реплику и выдать команду *Сервис, Репликация*, *Создать дополнительную реплику*. В открывшемся окне указывается имя и местоположение дополнительной реплики.

2. Проведите модификацию данных в основной и дополнительных репликах. Например, введите новую запись в основную реплику и удалите какую-либо запись из дополнительной; внесите изменения в полях основной и дополнительной реплик.

3. Выполните синхронизацию реплик.

Синхронизацию реплик можно выполнять при работе в Access или в среде Windows.

В первом случае нужно при открытой БД выполнить команду меню *Сервис*, *Репликация*, *Синхронизация*. В появившемся окне синхронизации можно определить место нахождения реплики, с которой требуется синхронизировать открытую БД, а также изменить статус одной из реплик набора.

Для изменения статуса синхронизируемых реплик достаточно отметить мышью флажок напротив предложенного системой варианта изменения статуса. К сожалению, пользователь не может задавать свои варианты изменения статуса.

При независимой работе пользователей с репликами могут возникать конфликты. Чаще всего они связаны с нарушением целостности данных (например, появление одинаковых значений в ключевом поле разных реплик) или с нарушением условий, контролирующих значения данных в таблице.

Если в процессе синхронизации система находит ошибки, пользователю об этом сообщается, и он устраняет их. Вызвать функцию контроля реплик на наличие конфликтов можно явно, задав команду из меню *Сервис, Репликация, Устранить конфликты.*

При обнаружении конфликтов Access выводит пользователю окно, содержащее информацию по устранению конфликтов. С его помощью можно выяснить причину и найти конфликтные записи в репликах, отобразить пояснение методов исправления ошибок.

4. Попытайтесь восстановить реплицированную базу данных до состояния нереплицированной. Для этого:

- создайте новую базу данных;

- методом импорта (*Файл, Внешние данные, Импорт*) перенесите в нее все таблицы;

- создайте запрос на каждую из таблиц с помощью конструктора. Выборка запроса должна содержать все поля, кроме системных (они, как правило, расположены в конце списка);

- в меню *Запрос* выполните команду, по которой выборка будет преобразована в таблицу базы данных (*Запрос на создание таблицы*);

- выполните этот запрос, в качестве имени новой таблицы укажите имя, которое будет отличаться от имени первоначальной реплицированной таблицы (например, с единицей в конце цепочки символов). После этого реплицированную таблицу можно удалить.

Лабораторная работа 2
Создание источника данных ODBC

Цель работы. Изучение технологии объектного связывания. Создание источника данных ODBC.

Архитектура ODBC предназначена для работы с базами данных реляционного типа, работа с которыми выполняется посредством языка SQL. ODBC включает следующие компоненты:

1. *Приложение*. Выполняет прикладные задачи, вызывает функции ODBC для передачи SQL-выражений и получения результатов. Приложение, использующее интерфейс ODBC, выполняет следующие задачи:

- запрашивает соединение (или сессию) с источником данных;
- посылает SQL- запросы к источнику данных;
- описывает область хранения и формат для результатов SQL- запросов;
- запрашивает данные;
- обрабатывает ошибки;
- если необходимо, оповещает пользователя об ошибках;
- осуществляет фиксацию или откат действий в режиме транзакций;
- закрывает соединение с источником данных.

2. *Диспетчер драйверов*. Диспетчер драйверов, поставляемый фирмой Microsoft®, является динамически подключаемой библиотекой (DLL). Основной задачей диспетчера является загрузка драйверов. Дополнительно он выполняет следующие функции:

- использует файл *odbc.ini* или системный реестр для установки соответствия между наименованием источника данных и DLL - драйвера;
- обрабатывает несколько инициализирующих вызовов ODBC;
- обеспечивает доступ ко всем функциям ODBC в каждом драйвере;
- проводит контроль параметров и последовательности вызовов функций ODBC.
- загружает драйвера по требованию приложения.

3. Источник данных. Содержит управляющую информацию, задаваемую пользователем. Информация источника данных используется интерфейсом ODBC для доступа к конкретной СУБД с помощью средств операционной системы и сетевой платформы. Источник данных − это понятие, объединяющее СУБД, операционную систему (ОС) и сеть.

4. Драйвер ODBC − это динамическая библиотека (DLL), которая реализует функции ODBC и взаимодействует с источником данных.

Диспетчер драйверов загружает ODBC-драйвер, когда приложение вызывает функцию SQLBrowseConnect, SQLConnect или SQLDriverConnect. Драйвер выполняет следующие функции в ответ на вызов приложением функции ODBC:

- устанавливает соединение с источником данных;
- передает запросы к источнику данных;
- преобразует данные из разных форматов (при необходимости);
- возвращает результат приложению;
- преобразует коды ошибок в стандартную форму и возвращает их приложению;
- описывает и манипулирует курсорами (эта операция скрыта от приложения, пока оно явно не требует доступа по имени курсора).

Задание к лабораторной работе 2

1. Создайте базу данных *Аэрофлот* согласно схеме (рис.1) и заполните ее исходными данными.

Схема БД состоит из четырех отношений:

− *Company* (ID_comp, name);

− *Trip*(trip_no, id_comp, plane, town_from, town_to, time_out, time_in);

− *Passenger*(ID_psg, name);

− *Pass_in_trip*(trip_no, date, ID_psg, place).

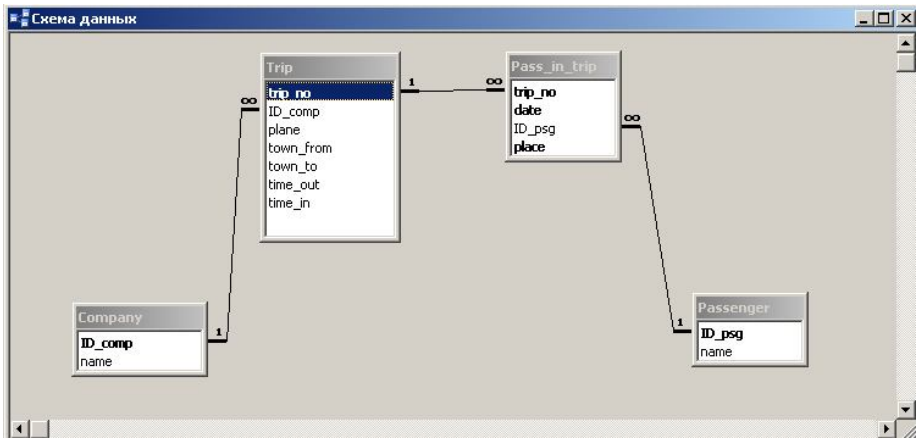

Рис.1. Схема базы данных *Аэрофлот*

Таблица *Company* содержит идентификатор и название компании, осуществляющей перевозку пассажиров.

Таблица *Trip* содержит информацию о рейсах: номер рейса, идентификатор компании, тип самолета, город отправления, город прибытия, время отправления и время прибытия.

Таблица *Passenger* содержит идентификатор и имя пассажира.

Таблица *Pass_in_trip* содержит информацию о полетах: номер рейса, дату вылета (день), идентификатор пассажира и место, на котором он сидел во время полета. При этом следует иметь в виду, что:

- рейсы выполняются ежедневно, а длительность полета любого рейса менее суток;

- время и дата учитываются относительно одного часового пояса;

- время отправления и прибытия указывается с точностью до минуты;

- среди пассажиров могут быть однофамильцы (одинаковые значения поля *name*);

- номер места в салоне – это число с буквой; число определяет номер ряда, буква (a – d) – место в ряду слева направо в алфавитном порядке.

2. Используя команду *Пуск, Настройка, Панель управления, Администрирование, Источники данных ODBC,* создайте источник данных *DSN* для этой базы данных.

Существует три типа источников данных ODBC:

- пользовательское DSN-имя, которое может применяться только одним пользователем и только на одном компьютере;
- системное DSN-имя, которое может использоваться всеми пользователями данного компьютера, кроме того, этот тип потребуется при создании приложения для работы с базой данных Internet;
- файловое DSN-имя, которое может быть скопировано и использовано для других компьютеров.

Для использования в дальнейших примерах создайте пользовательское DSN-имя.

Для его создания необходимо выполнить следующие действия:

- в диалоговом окне администратора источников данных (рис.2) активизируйте вкладку *Пользовательское DSN-имя*;
- щелкните по кнопке *Добавить*;
- в диалоговом окне нового источника данных щелкните на имени драйвера базы данных (в данном случае это Microsoft Access);
- щелкнув на кнопку *ОК*, в окне программы мастера создания нового источника данных укажите имя источника (оно должно быть запоминающимся, например, название AERO);
- по кнопке *Выбрать* укажите путь, где находится БД.

Это имя источника данных будет использоваться в приложении, написанном чуть позднее.

Пример. Подключение к базе данных с использованием провайдера ODBC с указанием DSN:

```
cn.Provider=”MSDASQL”
cn.ConnectionString = “DSN=ИМЯ”.
```

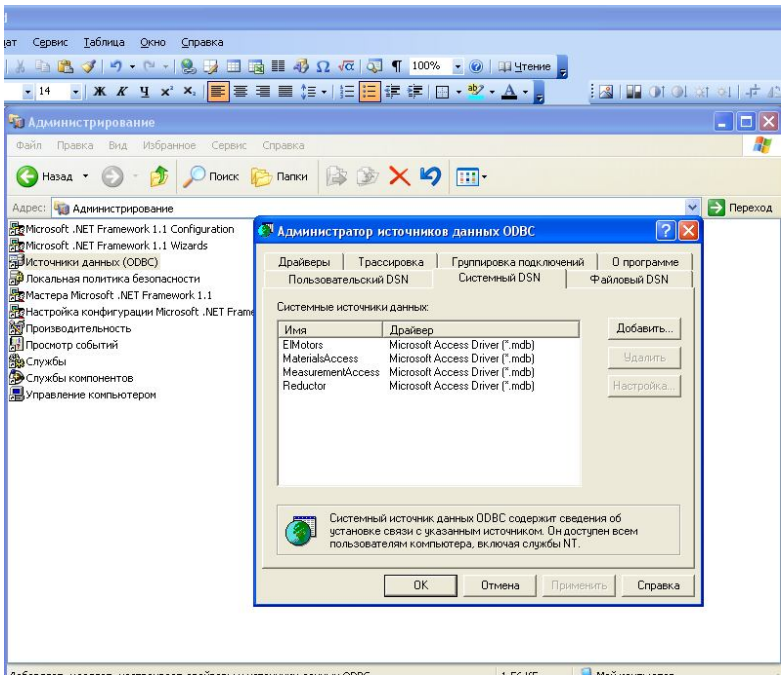

Рис.2. Диалоговое окно администратора источников данных

При использовании имени источника данных в строке подключения необходимо, чтобы имя *name1* действительно было зарегистрировано на компьютере клиента.

Лабораторная работа 3
Использование объектов ADODB

Цель работы. Технология объектного связывания баз данных. Использование объектов ADODB

Объект ADODB – это OLE - сервер, предназначенный для объектного связывания баз данных различных форматов в одном приложении. Иными словами, его интерфейс может использоваться приложением Access, любой программой, способной исполнять код на языке VBA (такой как Excel, Word), либо инструментальными средствами программирования сторонних пользователей (не Microsoft).

Рассмотрим некоторые объекты интерфейса ADODB.

Connection

Соединение (Connection) − блок информации, указывающий текущему приложению, как общаться с базой данных.

Объявление. Чтобы открыть новое соединение, вначале необходимо объявить объект класса Connection. Синтаксис:

Dim *ИмяСоединения* As New ADODB.Connection

Сразу после набора фразы ADODB, завершающейся символом точки, в окне редактора программного модуля всплывает контекстное меню, перечисляющее возможные продолжения конструкции. Если это не происходит, значит, в диалоговом окне *References*, вызываемой командой *TOOL, References*, не помечен флажком элемент *Microsoft ActiveX Data Object 2.1 Library* списка *Available References*.

Помимо создания строки подключения с учетом предварительного конфигурированного имени источника ODBC, можно также создать строку подключения, для которой вообще не нужно *DSN*-имя. Такая строка содержит всю необходимую для регистрации информацию, включая имя драйвера:

Driver={SQL Server};Server=name1;Database=Name;UID=olga;PWD=secret

Помните, что в данной строке важен не порядок размещения отдельных предложений, а их правописание. Наиболее распространенные ошибки – это пропуск символа (например, точки с запятой или фигурной скобки) или вставка лишнего (например, пробела до или после знака равенства).

Объявив объект соединения, вы можете затем открыть его.

Открытие. Команда открытия соединения предполагает наличие информации о содержимом так называемой строки соединения, имени пользователя, его пароле, значениях дополнительных параметров.

Указание провайдера и строки подключения

Указать провайдера данных можно с помощью свойства *Provider* объекта *Connection*. Если провайдер не указан, будет использован задаваемый по умолчению провайдер ODBC для OLE DB, который называется MSDASQL.

Свойство *Provider* объекта *Connection* – это текстовая строка, объявляющая тип провайдера OLE DB, который будет применен для подключения. При использовании провайдера ODBC для OLE DB его можно не указывать, так как он применяется по умолчанию. Однако для большей наглядности его все же рекомендуется указывать явным образом.

Строка подключения в ADO используется для предоставления информации о способе подключения к серверу базы данных. При использовании провайдера ODBC для OLE DB строка подключения будет такой же, что и строка подключения ODBC. Это означает, что точный вид информации, полученной провайдером ODBC, может варьироваться в зависимости от способа реализации строки подключения. Возможные аргументы строки подключения приведены в таблице. Причем, для разных провайдеров строка подключения может иметь совершенно разный синтаксис.

При использовании провайдера ODBC свойство ConnectionString может содержать имя источника данных *DSN*, это также может быть подключение без указания этого имени.

Строка подключения включает в себя сведения о производителе системы управления базой данных, сгруппированные в секции *Provider*. В этих данных отражается наименование, номер версии и другие особенности конкретной СУБД. Строка соединения с секцией *Provider*, представляет новейшую версию программного сервера *Jet* из состава Access 2000, выглядит так:

"Provider= Microsoft.Jet.OLEDB.4.0;Data Source=ПутьКБазеДанных”

Синтаксис строки подключения определяется провайдером.

Для отправки команд источнику данных в ADO следует установить к нему подключение с помощью метода *Open* объекта *Connection*. После окончания работы с источником данных его следует закрыть с помощью метода *Close* объекта *Connection*.

Таблица

Аргументы строки подключения

Аргумент	Описание
UID=	Учетная запись пользователя
PWD=	Пароль пользователя
DSN=	Имя источника данных, которое создано с помощью диспетчера драйверов ODBC Driver Manager
Driver=	Необходимый драйвер ODBC
Database=	Имя базы данных, к которой требуется создать подключение
APP=	Имя приложения, которое подключается к базе данных
Language=	Язык локализации, используемый сервером
Server=	Имя сервера SQL Server, к которому подключается приложение

Синтаксис метода *Open* объекта *Connection* (назначение аргументов приведено в таблице выше):

```
Cn. Open [Connect], [UserID], [Password]
```

Здесь Cn − имя соединения, прописанное в операторе Dim (см. стр. 14).

Все перечисленные аргументы этого метода не являются обязательными. Например, строку подключения можно указать, используя не аргумент Connect этого метода, а значение свойства ConnectionString объекта Connection. Результат в обоих случаях будет одинаковым.

Соединение с базой данных Access может быть открыто даже в том случае, если строка соединения не содержит имени пользователя и его пароля. В этом случае будут использованы значения, принятые по умолчанию.

Например: `Cn. Open “DSN=name1”`

или: `Cn. Open`

Закрытие

По окончании работы с объектом Connection модели объекта ADO следует всегда закрывать его с помощью метода Close согласно приведенному ниже синтаксису:

```
cn.Close
```

Закрытие подключения к источнику данных явным образом позволяет регулярно высвобождать связанные с этим подключением ресурсы (либо на стороне клиента, либо сервера, либо сразу на обеих).

Пример 1. Подключение к базе данных с использованием провайдера ODBC с указанием DSN:

```
Dim Cn. As New ADODB. Connection
cn.Provider=”MSDASQL”
cn.ConnectionString=“DSN=name1”
Cn. Open
```

Пример 2. Подключение без указания имени источника данных DSN, но с использованием технологии ODBC:

```
Dim Cn. As New ADODB. Connection
cn.Provider=”MSDASQL”
cn.ConnectionString=”Driver={SQL Server}; DataBase=Name1;UID=randy; PWD=prince;”
Cn. Open
```

Такое подключение к серверу будет установлено быстрее, так как для него не потребуется считывать DSN-информацию из системного реестра Windows. Однако оно менее гибкое, потому что информация о подключении «намертво» зафиксирована в откомпилированном коде.

Пример 3. Подключение *Jet-провайдера* для OLE DB объекта *Connection* (для связи с СУБД Access). Для *Jet-провайдера* строка подключения состоит из пути и имени .mdb-файла:

```
Dim Cn. As New ADODB. Connection
cn.Provider =”Microsoft.Jet.OLEDB.3.51”
cn.ConnectionString = ”c:\data\name1.mdb”
cn. Properties(“Jet OLEDB : System database”) = ”c:\data\name1.mdw”
cn. Properties (“Password”) = ”mypass”
cn. Properties (“User ID”) = “myname”
```

Пример 4. Подключение к базе данных SQL Server осуществляется с использованием провайдера SQLOLEDB1 с помощью приведенного ниже синтаксиса:

```
Dim Cn. As New ADODB. Connection
cn.Provider=”SQLOLEDB1”
```

cn.ConnectionString = “database=mydata; Server = mysrv; UID = User; PWD = mypass”

Пример 5. Использование OLE DB.Connection:

```
Sub Primer1()
Dim Connection As New ADODB.Connection
Connection.Open("Provider= Microsoft.Jet.OLEDB.4.0;”  & _
”Data Source=C:\Data\Basa.mdb”
        {код программы}
Connection.Close
End Sub
```

Объект Recordset

После объявления объекта Connection и успешного открытия соединения будет естественным обратиться к объекту Recordset, задающему некий набор данных и позволяющему приступить к решению конкретной задачи. Объект класса Recordset способен содержать данные из таблицы, запроса, хранимой процедуры. Открытый объект Recordset может единовременно ссылаться на одну запись (строку) набора данных.

Открытие. Чтобы открыть объект New ADODB, следует объявить переменную *ИмяНабораДанных* типа Recordset, а затем записать конструкцию вида:

ИмяНаборДанных.Open *ИмяОбъектаДанных*, *ОбъектСоединения*, *ТипКурсора*, *ТипБлокировки*, *ДополнительныеОпции.*

ИмяНаборДанных − это ранее объявленная переменная типа ADODB.Recordset. В качестве *ИмяОбъектаДанных* следует задать имя таблицы, курсора, либо символьной переменной, содержащей код SQL. Вместо *ОбъектСоединения* можно подставить имя объекта *CurrentProject.Connection*, если достаточно использовать ту же базу данных, которая содержит текст текущего модуля, либо другого правильно определенного объекта типа *Connection*.

Свойство *CursorType* (*ТипКурсора*) определяет, как вы передвигаетесь по набору записей, оно обязательно при использовании набора записей с

формами. Свойство *LockType* (*ТипБлокировки*) работает вместе со свойством *CursorType* и контролирует блокировку использования записи кем-то другим. Блокировка может использоваться для разрешения конфликтов редактирования в многопользовательской среде.

Параметры *ТипКурсора* и *ТипБлокировки* принимают значения перечислимых типов CursorTypeEnum, LockTypeEnum, которые подробно описаны в лабораторной работе 5.

Закрытие. При завершении работы с набором данных его необходимо закрыть инструкцией

ИмяПеременной.Close.

Редактирование записи. Команды добавления записи и редактирования ее содержимого выполняются с помощью следующих инструкций:

- *ИмяПеременной*.AddNew (добавление записи);
- *ИмяПеременной*.Delete (удаление записи);
- *ИмяПеременной*.Update (сохранение внесенных изменений).

Повторяю, если вам достаточно ссылаться только на наборы данных в текущей базе, в которой хранится программный код, используйте объект *CurrentProject.Connection.*

Пример 6. Открытие набора данных, настроенного на таблицу с именем *Tabl* в текущем соединении. В примере выполняется добавление записи с именами полей *Name* и *Family*, *Таб_номер*. Здесь *Таб_номер* является первичным ключом (текстовое значение).

```
Sub Primer6( )
Dim RS As New ADODB. Recordset
RS.Open “Tabl”, CurrentProject.Connection, adOpenKeyset, adLockOptimistic
RS.Addnew
RS («Таб_номер»). Value= «0010»
RS(“name”).Value=”Olqa”
RS(“Family”).Value=”Karaseva”
RS.Update
```

```
RS.Close
Set RS=Nothing
Connection.Close
    End Sub
```

Условия достижения начала и конца набора данных. Объект класса ADODB.Recordset содержит два метода определения факта достижения начальной и конечной записей набора данных – соответственно BOF и EOF.

Перемещение по записям. Класс ADODB.Recordset содержит ряд функций, позволяющих перемещаться по записям открытого набора данных − Move, MoveFirst, MoveLast, MoveNext и MovePreviouns, что соответствует значениям смещения указателя текущей записи: на произвольное число записей относительно текущей, к первой записи, последней записи, следующей записи, предыдущей записи.

Использование полей. Набор данных способен единовременно ссылаться только на одну текущую запись. Доступ к свойству Value класса *Field* можно увидеть в приведенном ниже примере 7.

Пример 7. Просмотр записей в текущей базе данных.

```
Sub prim( )
  Dim rs As New ADODB.Recordset
rs.Open " Tabl ", CurrentProject.Connection, adOpenKeyset, _
                    adLockOptimistic
Dim p As String
rs.MoveFirst
Do While rs.EOF = False
  p = rs("Name").Value
  Msgbox p
  rs.MoveNext
Loop
rs.Close
Set rs = Nothing
End Sub
```

Пример 8. Просмотр записей в закрытой базе данных *Аэрофлот* с использованием источника данных ODBC с именем *AVTO*. Предполагается, что в базе данных, для которой создан источник ODBC, существует таблица *Клиенты*. В таблице *Клиенты* есть поле *Фамилия.*

В режиме пошаговой отладки (*Debug, F8*) можно прочитать значения промежуточных данных.

```
Sub prim1()
Dim con As New ADODB.Connection
con.Provider = "MSDASQL"
con.ConnectionString = "DSN=AVTO"
con.Open
Dim rs As New ADODB.Recordset
rs.Open "Клиенты", con
Dim p As String
rs.MoveFirst
Do While rs.EOF = False
  p = rs("Фамилия").Value
  MsgBox p
  rs.MoveNext
Loop
rs.Close
con.Close
Set con = Nothing
Set rs = Nothing
End Sub
```

Использование объектов ADOX

Библиотека ADOX содержит определения классов, позволяющих управлять таблицами, курсорами, индексами, а также администрировать данные о пользователях и группах пользователей.

Чтобы получить доступ к библиотеке ADOX, откройте окно редактора VBA и выберите в строке меню команду *TOOLS, References*. Пролистайте список окна *References* и установите флажок для элемента Microsoft ADO Ext.2.1 for DLL and Security, а затем щелкните на кнопке OK.

Объект Catalog

Каталог (Catalog) – это контейнерный класс для хранения данных о таблицах, курсорах, хранимых процедурах, пользователях и группах.

Между объектами Catalog и Connection существует связь типа один-к-одному. Если необходимо создать с помощью программного кода таблицу и добавить ее в базу данных, определить новые или исправить существующие ключи и индексы, ввести или отредактировать атрибуты пользователей или групп, все эти операции вы сможете сделать с помощью объекта Catalog.

Пример процедуры поиска данных приведен в примере 9. В этой программе выполняется поиск записей таблицы *Клиенты*, содержащей в каком-либо из полей значение, введенное с помощью диалогового окна в переменную *Temp*. В случае, если такая запись найдена, на экран помещается сообщение, содержащее искомое значение и значение ключевого поля, соответствующего найденной записи (*ID*).

Пример 9. Поиск данных в таблице *Tabl* по образцу, введенному с экрана оператором *InputBox.* В режиме пошаговой отладки (Debug, F8) можно прочитать значения промежуточных данных. Здесь Таб_номер − ключ таблицы.

```
    Sub findData( )
  Dim Connection As New ADODB.Connection
  Dim Catalog As New ADOX.Catalog
  Dim Recordset New ADODB.Recordset
  Connection.Open("Provider= Microsoft.Jet.OLEDB.4.0;"  & _
  ”Data Source=C:\Data\Basa.mdb”
  Set Catalog.ActiveConnection = Connection
  Recordset.Open “Tabl”, Catalog.ActiveConnection, adOpenDynamic, _
  adLockOptimistic
  Recordset.Fields.Refresh
  Dim Temp As String
```

```
Recordset.MoveFirst
Temp = InputBox("Введите данные для поиска  (Q – выход):", _
                                    "Поиск данных")
Do While (RS.EOF=false)
    If (Temp = Q) Then Exit Do
        For Each Field In Recordset.Fields
            If (Field.Value Like Temp) Then
            MsgBox "Найдено: " & Field.Value & " в " &  _
            RecordSet ("Таб_номер").Value
            Exit For
            End If
        Next
Recordset MoveNext
Loop
Recordset.Close
Set Recordset =Nothing
Set Catalog =Nothing
Connection.Close
Set Connection =Nothing
    End Sub
```

Задание к лабораторной работе 3

1. Убедитесь в том, что в вашем распоряжении находится база данных СУБД Access на локальном компьютере (например, *Аэрофлот*).

2. Создайте «пустую» базу данных, которая будет использоваться в качестве проекта для подключения к удаленной базе данных.

3. Создайте источник данных ODBC подключения к базе данных по описанному в предыдущей работе алгоритму.

4. Откройте проект, создайте модуль и в редакторе напишите код процедуры подключения ко второй базе данных, используя источник данных ODBC, настроив имена на свой вариант (пример приведен).

5. Создайте модуль и в редакторе напишите код процедуры подключения к текущей (открытой) базе данных, настроив имена на свой вариант (пример приведен).

6. Создайте еще один код, обратитесь к базе данных, написав строку подсоединения, и выполните поиск данных по этой базе данных, настроив имена на свой вариант (пример приведен).

Лабораторная работа 4
Запросы в кодах VBA

Цель работы. Запросы на языке SQL в кодах VBA

Существуют следующие способы выполнения запросов, которые приведены ниже

1. Вызов метода Execute *(*для выполнения запросов SQL на изменение).

2. Создание и выполнение специального объекта QueryDef.

3. Использование инструкции SQL в качестве аргумента метода OpenRecordset.

4. Выполнение метода OpenRecordset для существующего объекта QueryDef.

5. Вызов методов RunSQL и OpenQuery.

Рассмотрим применение этих способов.

1. Метод Execute используется, если требуется выполнить такое изменение в БД, при котором не возвращаются записи. Это, например, операции вставки или удаления записей. В качестве простейшего примера приведем команды Visual Basic для приложений выполнения запроса на изменение, в котором выполняется обновление записей таблицы *Pass_in_trip*, не имеющих значение в столбце *place*. Обращаться будем к удаленной базе данных, используя технологию DAO. Для этого подключите библиотеку MS DAO 3.51 (Tools, Reference). База данных называется *AERO*. Параметр dbFailOnError означает, что при возникновении ошибки все изменения отменяются.

```
Sub Prim1_ Execute ()
    Dim dbsAERO As DAO.Database
    Set dbsAERO = OpenDatabase("C:\AERO.mdb")
    Dim strSQL as String
    strSQL =”DELETE FROM Pass_in_trip WHERE place IS NULL”
```

```
    dbsAERO.Execute strSQL, dbFailOnError
    dbs.Close
End Sub
```

2. Объект QueryDef представляет собой сохраненное определение запроса в базе данных. Его можно рассматривать как откомпилированную инструкцию SQL. Приведенная ниже программа выполняет создание нового объекта QueryDef. Обращаться будем к удаленной базе данных, используя технологию DAO. Предложение dbs.QueryDefs.Delete ("Полеты из Екатеринбурга ") означает, что следует удалить уже существующий запрос с этим именем.

```
Sub Prim2_ QueryDef  ()
    Dim strSQL As String
    Dim qdf As DAO.QueryDef
    Dim dbs As DAO.Database
    Dim rst As DAO.Recordset
    Set dbs = OpenDatabase("C:\AERO.mdb")
    Set rst = dbs.OpenRecordset("Trip")
    strSQL = "Select * From Trip where town_from='Екатеринбург' "
    dbs.QueryDefs.Delete ("Полеты из Екатеринбурга ")
    Set qdf = dbs.CreateQueryDef("Полеты из Екатеринбурга", strSQL)
    MsgBox ("Запрос на полеты из Екатеринбурга готов")
    dbs.Close
End Sub
```

3. Метод OpenRecordset используется, чтобы открыть объект типа Recordset для выполнения последующих операций над ним.

В следующей процедуре с помощью инструкции SQL создается объект Recordset типа динамического набора записей. В предложение WHERE инструкции SQL включена функция *Year*, определяющая отбор номеров рейсов, выполненных в 2006 году. Метод RecordCount возвращает количество записей, удовлетворяющих условию. Обратите внимание, работа выполняется в текущей базе данных.

```
Sub Prim3_OpenRecordset()
Dim dbs As DAO.Database
Dim rst As DAO.Recordset
Dim strSQL As String
Set dbs = CurrentDb
strSQL = "SELECT DISTINCTROW trip_no, date FROM Pass_in_trip WHERE
(Year(date)=2006)"
 Set rst = dbs.OpenRecordset(strSQL, dbOpenDynaset)
 rst.MoveLast
'Debug.Print rst.RecordCount
kol = rst.RecordCount
MsgBox "Количество найденных записей = " & kol
End Sub
```

Внимание. Сообщение «Слишком мало параметров: требуется 1» может означать, что в запросе присутствуют неверные имена полей.

4. В следующем примере запрос *Список-пассажиров* (он должен уже существовать) открывается в режиме таблицы, в котором пользователю разрешается просмотр записей.

```
Sub Prim4_ OpenQuery()
 Dim dbs As DAO.Database
 Set dbs = CurrentDb
DoCmd.OpenQuery «Список-пассажиров», acReadOnly
 End Sub
```

Выбор варианта выполнения запросов определяется программистом с учетом особенностей решаемой задачи.

5.1. Метод RunSQL выполняет макрокоманду *ЗапускЗапросаSQL* (*RunSQL*) в программе VBA. В следующем примере изменяется название города (*town-to*) для всех номеров рейсов в таблице *Trip*. Объект DoCmd доступен только для текущей БД!

```
Sub Prim4_ RunSQL ()
Dim dbs As DAO.Database
```

```
Set dbs = CurrentDb
DoCmd.RunSQL “UPDATE Trip SET town-to = ‘Rostov-na-donu’ “ &
“WHERE town-to = ‘Rostov’;”
End Sub
```

5.2. Метод OpenQuery выполняет макрокоманду *ОткрытьЗапрос* (OpenQuery) в программе VBA. С его помощью можно открыть запрос в режиме таблицы, конструктора или просмотра. При этом устанавливается один из следующих режимов работы с данными: добавление, изменение или только чтение.

Задание к лабораторной работе 4

Создайте проект подключения к базе данных *Аэрофлот*, включающий в себя модули:

- использующие запросы (метод Execute**);**
- на создание и выполнение специального объекта QueryDef;
- использующие запросы (метод OpenRecordset**);**
- использующие запросы (метод OpenRecordset для существующего объекта QueryDef);
- использующие запросы (метод RunSQL)
- использующие запросы (метод OpenQuery).

Лабораторная работа 5
Интерфейс к удаленной БД

Цель работы. Создание интерфейса к удаленной базе данных.

Динамическое присвоение наборов записей

В лабораторной работе 4 обсуждалось, как использовать SQL для создания набора записей в среде VBA. Наборы записей собираются и хранятся в объекте Recordset. В действительности набор записей содержит совокупность указателей на данные, требующиеся объекту Recordset.

Начиная с Access 2000, вы можете динамически присваивать форме эти наборы записей, используя VBA. Однако это несколько отличается от встраивания кода SQL в форму, как вы делали (через источник записей).

Предположим, что теперь вы хотите, чтобы ваш набор записей содержал только те рейсы, которые вылетают в Москву (*town-to* = ‘Moscow’). Более того, вы хотите, чтобы ваша форма (форма *Рейсы*) выводила на экран этот набор записей.

Возможно, вам будет проще, особенно если у вас большой код и длинное имя формы, присвоить имя формы строковой переменной и использовать ее вместо полного имени формы.

Пример 1. Обращение к базе данных с помощью формы, имеющей в коде псевдоимя *strFrmNm*.

```
Sub runFormNY()
Dim con As ADODB.Connection Dim recSet As Recordset
Dim strFrmNm As String
Set recSet = New ADODB.Recordset
recSet. CursorType = adOpenKeyset
recSet.LockType = adLockOptimistic
Set con = New ADODB.Connection
con.Open "Provider=Microsoft. Jet .OLEDB. 4 . 0; Data  Source=C:\Аэрофлот.mdb;"
recSet. Open "SELECT * FROM Trip WHERE  town_to = ‘Moscow’; ",con
strFrmNm = "Рейсы"
DoCmd.OpenForm strFrmNm
Set Application.Forms(strFrmNm).Recordset = recSet
recSet.Close
con.Close
Set recSet = Nothing
Set con = Nothing
End Sub
```

Мы создали набор записей. Обсудим назначение двух нижеперечисленных строк:

```
recSet.CursorType = adOpenKeyset
recSet.LockType = adLockOptimistic
```

Свойство *CursorType* (Тип курсора) определяет, как вы передвигаетесь по набору записей, оно обязательно при использовании набора записей с формами. По существу, есть четыре основных типа курсоров.

1. Dynamic (adOpenDynamic) − динамический курсор. Это означает, что изменения, вносимые в исходные данные, динамически включаются в набор записей. Поэтому, если кто-то другой изменяет источник данных, вы сможете увидеть эти изменения. Вы также свободно можете вносить необходимые изменения в набор записей и передвигаться по набору в любом нужном вам направлении.

2. Keyset (adOpenKeyset) − курсор набора данных. Этот курсор не позволяет набору записи динамически включать изменения, внесенные другими в источник данных. Однако вы можете редактировать набор записей по своему усмотрению.

3. Static (adOpenStatic) − статический курсор. Не допускается вносить изменения в набор записей. Вы можете просматривать его, двигаясь вперед или назад.

4. Forward-only (adOpenForwardOnly) − курсор, предназначенный для последовательного перемещения по набору записей только вперед. Вы не можете вносить изменения или переходить назад в наборе записей. Вы можете лишь двигаться вперед. Вышеприведенный список систематизирован по степени производительности. Чем больше взаимодействие с набором записей, тем медленнее исполнение. Dynamic − самый медленный, а *Forward-only* − самый быстрый. Если вы работаете с большим объемом данных, вам потребуется принимать решение, какой тип курсора (*CursorType*) будет работать лучше всего и при этом сохранит эффективность.

Во второй новой строке используется свойство *LockType* (Тип блокировки). Оно работает вместе со свойством *CursorType* и контролирует блокировку использования записи кем-то другим. Блокировка может использоваться для разрешения конфликтов редактирования в многопользовательской среде.

Есть четыре типа блокировки:

- *adLockReadOnly* − только чтение. При редактировании блокируется весь набор записей;

- *adLockPessimistic* − пессимистическая блокировка. На время редактирования блокируется та запись, которая редактируется;

- *adLockOptimistic* − оптимистическая блокировка. Она предупреждает конфликты, блокируя запись, только во время сохранения внесенных изменений. Это позволяет избежать ситуации, когда два человека сохраняют измененную запись в одно и то же время;

- *adLockBatchOptimistic* − пакетная блокировка. Она используется, когда записи обновляются в пакетном режиме.

Добавление и редактирование данных

Если вы запускали процедуру из предыдущего раздела (Sub runFormNY()), то, должно быть, заметили интересную проблему: блокировку добавления новых данных в базу данных, с которой вы соединены. В этом случае вы не сможете отредактировать данные. Обратите внимание, что кнопка *New Record* (*Новая запись*) на форме затенена.

Кроме свойства CursorType, есть еще и другое свойство курсора, с которым вы будете иметь дело: CursorLocation (*Положение курсора*). Оно определяет местонахождение (на стороне клиента или на стороне сервера) службы, которая отслеживает движение указателя записи по набору данных. По умолчанию, без задания положения курсора CursorLocation, записи открываются в режиме «только для чтения». Однако вы можете исправить это, добавив следующую выделенную строку:

```
Set recSet = New ADODB.Recordset
recSet.CursorType   - adOpenKeyset
recSet.LackType   =  adLofckOptimistic
recSet.CursorLocation = adUseClient
```

Кнопка *New Record* (*Новая запись*) теперь доступна, как и возможность полного редактирования.

Контроль над элементами управления в формах

Каждый объект на форме называется элементом управления. Все, что у нас здесь есть, − это надписи и текстовые поля**.** Поля связаны со столбцами таблицы *Trip* или в зависимости от вашего кода с набором записей. При автоматическом создании формы Access присваивает имена этим элементам управления автоматически.

Лучше изменить имена в окне свойств формы, чтобы можно было отличить имя поля таблицы от имени управляющего элемента, связанного с ним. Например, имя поля – *trip_no*, а имя управляющего элемента на форме – *ctl_trip_no*. Это следует выполнить в режиме конструктора формы в свойстве *Имя* (рис. 3).

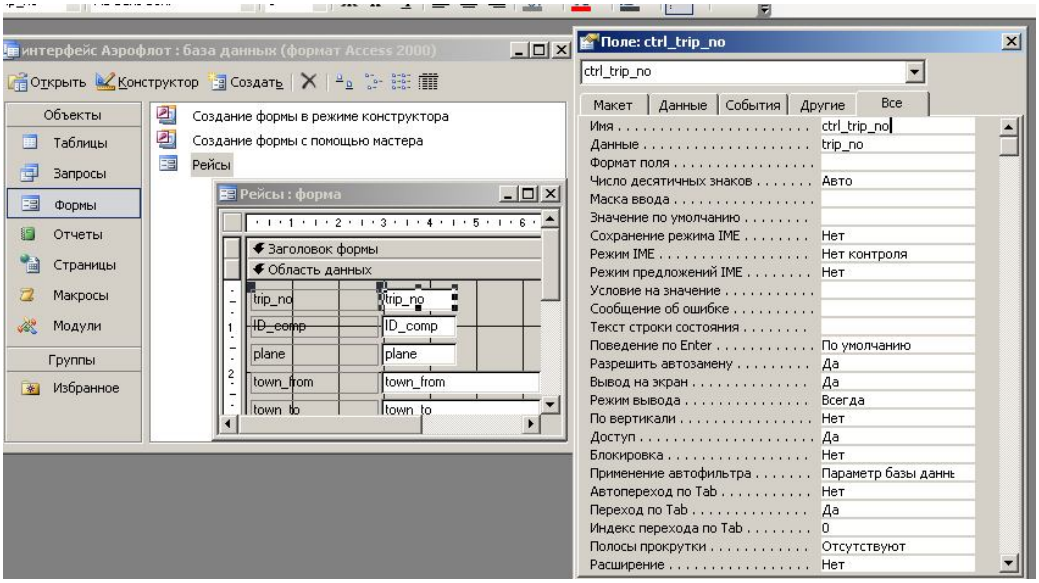

Рис. 3. Окно свойств: имя управляющего элемента в проекте отличается от имени поля, которое будет ему соответствовать

Access 2003 работает с некоторыми аспектами элементов управления форм иначе, чем версии 2000 и 2001. Поэтому те техники, которые показаны, может, и не являются самыми элегантными решениями, но они работают во всех версиях и довольно просты в использовании.

Посмотрим на следующий блок кода (пример 2). Ниже приведены пояснения к этому коду.

Пример 2

```
Sub runFormAll()
Dim con As ADODB.Connection
Dim recSet As Recordset
Dim strFrmNm As String
Set recSet = New ADODB.Recordset
recSet.CursorType = adOpenKeyset
recSet.LockType = adLockOptimistic
recSet.CursorLocation = adUseClient
Set con = New ADODB.Connection
con.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Аэрофлот.mdb;"
recSet.Open "SELECT * FROM Trip", con
strFrmNm = "Рейсы"
DoCmd.OpenForm strFrmNm
Set Application.Forms(strFrmNm).Recordset = recSet
Form_ Рейсы.ctrl_trip_no.SetFocus
Form_ Рейсы.ctrl_ID_comp.Visible = False
recSet.Close
con.Close
Set recSet = Nothing
Set con = Nothing
End Sub
```

Рассмотрим смысл записи *Form_ Рейсы*.

Создадим в режиме конструктора (Свойства) код на обработку события *Открытие* (рис. 4). После этого в окне проекта появится модуль формы (*Form_Рейсы*). Эти манипуляции нужны для того, чтобы имя формы появилось в семействе.

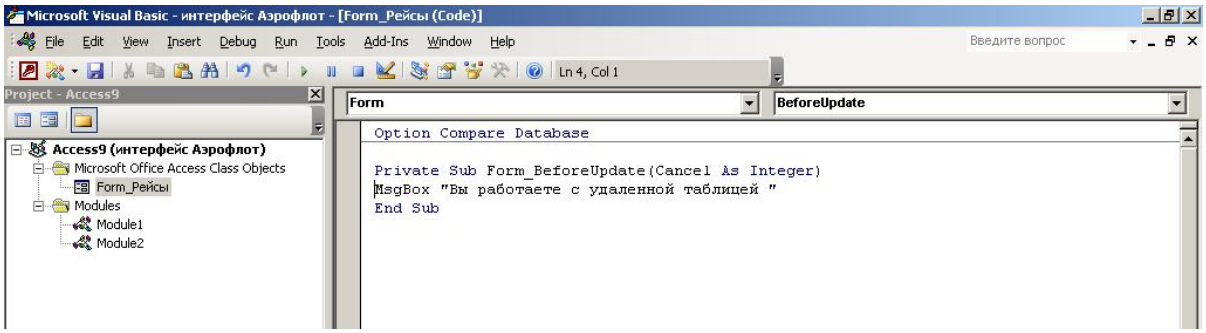

Рис.4. Модуль формы «Рейсы»

Теперь в окне базы данных во время открытия формы вы увидите диалоговое окно (рис.5), но главное − это появление имени Form_ Рейсы. (рис. 4).

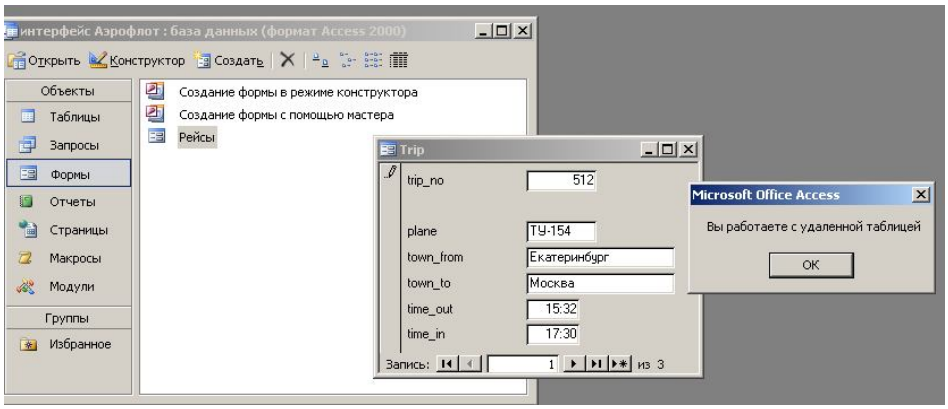

Рис.5. Результаты выполнения кода (пример 2)

Появившийся модуль формы позволит легко обратиться к любому управляющему элементу формы и задать ему необходимые свойства (будет предлагаться соответствующий список во время набора кода).

Обратите особое внимание на следующую строку:

```
Form_ Рейсы.ctrl_ID_comp.Visible = False
```

Результатом выполнения этой строки является скрытое поле *ID_comp*. Как вариант, вы могли изменить свойство с помощью двух следующих строк:

```
Form_ Рейсы. ctrl_ID_comp.Visible = True
Form_ Рейсы. ctrl_ID_comp.Visible = False
```

По существу, это сделает *ctlCustNumber* доступным только для чтения, не скрывая его совсем.

Для каждого элемента можно назначить список свойств, которые используются при необходимости. Нередко используется конструкция *If … Then*: если наступает условие А, необходимо использовать одну установку свойств, если наступает условие В – другую.

Важно обратить внимание, что вы можете делать условное форматирование. Например, если число отрицательное, вы можете изменить его цвет на красный.

Задание к лабораторной работе 5

1. Подготовьте к работе базу данных (вариант – из предложенных преподавателем).

2. Расположите ее в папке.

3. Сделайте форму к таблице базы данных для ввода, редактирования данных.

4. Создайте новый проект, не включающий ни одну из таблиц (можно на рабочем столе).

5. Импортируйте в этот проект созданную в п. 3 форму (только форму).

6. В режиме конструктора удалите для импортированной формы источник записей (*Конструктор, Свойства,* вкладка *Данные, Источник записей*).

7. На вкладке *Модули* создайте код для просмотра записей таблицы базы данных, находящейся в папке, установив соединение. Проверьте, можно ли вводить новые и редактировать существующие записи. Объясните.

8. Создайте вторую версию кода, позволяющего редактировать и вводить данные.

Рекомендуемая литература

1. Чарльз Е. Браун. Access VBA. Программирование в примерах / Чарльз Е. Браун, Рон Петруша. М.: Кудиц-образ, 2006. 432 с.

2. Хомоненко, А.Д. Базы данных: учебник / А.Д. Хомоненко, В.М. Цыганов. М.: Компьютерная книга, 2006. 735 с.

Оглавление

Введение 3
Лабораторная работа 1. Технология реплицирования данных..... 3
Лабораторная работа 2. Создание источника данных ODBC 9
Лабораторная работа 3. Использование объектов ADODB 13
Лабораторная работа 4. Запросы в кодах VBA 24
Лабораторная работа 5. Интерфейс к удаленной БД 27
Рекомендуемая литература……...35